Multi-Class Instance-Incremental Framework for Classification in Fully Dynamic Graphs

Ritvik Shrivastava

Department of Information Technology
Netaji Subhas Institute of Technology, University of Delhi
Delhi, India
E-mail: ritviks.it@nsit.net.in

Sreyashi Nag

Department of Instrumentation & Control Engineering
Netaji Subhas Institute of Technology, University of Delhi
Delhi, India
E-mail: sreyashin.ic@nsit.net.in

Hardeo Kumar Thakur

Department of Computer Science and Technology
Manav Rachna University
Faridabad, India
E-mail: hkthakur@mru.edu.in

Anand Gupta

Department of Computer Engineering
Netaji Subhas Institute of Technology, University of Delhi
Delhi, India
E-mail: omaranand@nsitonline.in

Abstract: Existing work in the area of graph classification is mostly restricted to static graphs. These static classification models prove ineffective in several real life scenarios that require an approach capable of handling data of a dynamic nature. Further, the limited work in the domain of dynamic graphs mainly focuses on solely incremental graphs which fail to accommodate Fully Dynamic Graphs (FDG). Hence, in this paper, we propose a comprehensive framework targeting multi-class classification in fully dynamic graphs by utilizing the efficient Weisfeiler-Lehman graph kernel (W-L) with a multi-class Support Vector Machine (SVM). The framework iterates through each update using the Instance-Incremental method while retaining all historic data in order to ensure higher accuracy. Reliable validation metrics are utilized for the model parameter selection and output verification. Experimental results over four case studies on real-world data demonstrate the efficacy of our approach.

Keywords: Fully Dynamic Graph; Dynamic Graph; Graph Classification; Multi-Class Classification.

1 Introduction

In today's vastly connected world, information is being exchanged at an exponential rate. Information in the form of networked data has become a ubiquitous entity on the World Wide Web due to the development and growth of social networking sites. The substantial degree of information handled by networked data has thus made it an increasingly popular source of data collection for the Big Data community. One of the main dimensions of big data is the large volume of information that needs to be processed and analyzed (Zikopoulos et al., 2011). Moreover, many of the real world problems involving big data consist of data that are inter-related. The magnitude of this data and the inter-related nature of the individual data elements have resulted in dynamic graphs becoming an increasingly apt form of representation for such data. This has thereby allowed the graphical form of representation to be frequently used for modeling a large number of real world problems. The relevance of this field of study is evident by the sheer variety of applications it finds its

usage in. Problems ranging from Text Categorization (Rousseau et al., 2015), Image Classification (Harchaoui and Bach, 2007) and online social networks (Wang et al., 2011, Han et al., 2017) to large scale biological systems and bioinformatics(Pavlopoulos et al., 2011, Rytsareva et al., 2014, Boulif and Atif, 2015) can all be essentially modeled as networked data and consequently, as graph classification problems.

By dynamic graphs, also referred to as 'fully dynamic graphs', we refer to graphs subject to regular updates - namely addition, deletion and modification of nodes (Demetrescu et al., 2004). Dynamic Graphs receive data in the form of continuous streams and hence keep changing in appearance and structure. The transient nature of such graphs can be utilized in a variety of applications involving the management of massive datasets by utilizing the associated networked structure of the data and examining the interrelations between them. The use of dynamic graphs can also be seen in applications like modeling communication lines between various cities in a region, the analysis and mining of user data by business enterprises and to observe the browsing history of a user, to name a few. One of the most pertinent uses of such dynamic graphs lies in the domain of graph classification. Social media networks such as Twitter and Facebook as well as sites like Netflix and Amazon are prime examples of graph classification techniques being used for recommendation algorithms. The growing demand for such applications has led to an increasing need for developing large-scale, robust algorithms that can be used to accurately classify such dynamic graphs.

The graph classification problem, essentially a supervised learning problem, has been vastly researched upon in the domain of static graphs (Jin et al., 2010, Kong and Philip, 2014, Zhao et al., 2011) in the past and has become the basis for future research in the field of dynamic graphs. Graph classification algorithms in dynamic graphs aim to form a classifier using a training data set and to use it for further label prediction of the data being dynamically added. The problem consists of extracting discriminatory features from nodes, subgraphs or the entire graph as a whole and using them to learn the prediction of nodes or edge labels in the graphs, depending on their specifications. Conventional approaches to graph classification involve extracting subgraphs, subtrees and walks and other similar substructures (Gago-Alonso et al., 2013, Gärtner et al., 2003, Shervashidze and Borgwardt, 2009), mining them and using the results to compute similarities between graphs. This is commonly done by using the widely popular kernel methods (Shervashidze et al., 2009). Graph Kernels (Vishwanathan et al., 2010) are tools of intensive use in this field for measuring similarity between graphs and making the graph data usable for kernel methods such as the Support Vector Machines (Joachims et al., 2009). A large amount of research has already been done on binary classification approaches to graph classification where labels of only +1 and -1 are available for the two classes (Jin et al., 2009). For static graphs, some of these algorithms can be naturally extended to multi-class classification while others may need additional modifications in order to include such an application. (Aly, 2005) describes various approaches of multi-class classification. In numerous real-world applications however, a dynamic graph is required to be classified into a multi-class structure. It is thus beneficial to explore multi-class classification approaches aiming to classify dynamic datasets. The following example serves to highlight the need for an algorithm that can accommodate multi-class classification in a fully dynamic graph.

Let us consider a real-world network formed by an e-commerce company, ABC. The company wants to classify the products available through its portal according to the various categories they belong to. Let each item up for sale on the website be a major node in the graph. The products on sale have been produced by different companies and are being sold on the website by various sellers. The companies and sellers form the minor nodes of the graph. A major node is connected to a minor node if the corresponding product is manufactured by that particular company or is being sold by that specific seller. Thus, this forms a graph with any two products on ABC (major nodes) connected to each other via a common manufacturing company or seller (minor nodes). The 'x' numbers of categories constitute the class labels for each major node. Our task is to predict which category a product is likely to be under and hence, this essentially becomes a graph classification problem.

This is a clear example of a situation where a multi-class classification model is in demand. The number of categories are likely to be more than two (books, clothing, electronics, etc.) and using a binary classification approach in this case would prove to be inadequate. Secondly, a situation can also arise where a manufacturer stops the production of a particular product and it is now being sold by a different seller. The ideal classifier would be one that can accommodate such changes in the graph. Hence, the aim of this paper is to construct a multi-class classifier model that can accommodate the deletion and modification of old nodes along with addition of new nodes.

This paper presents a framework which is able to address both of the above situations, namely that of multi-class classification in fully dynamic graphs. We take inspiration from the existing work of (Yao and Holder, 2014) which has provided an impressive approach combining Support Vector Machines (SVM) with the W-L Kernel to classify nodes into binary classes in dynamically incremental graphs. Although their work

is unique in this area, it has still shown some scope for improvement due to the following reasons:

- It discards old batches of nodes and edges along with the corresponding support vectors which could lead to loss in accuracy over time.
- It only targets graphs that are solely incremental in nature.
- It only deals with binary classification.
- It incorporates the batch-based incremental learning approach which has a few limitations (Read et al., 2012) when applied to a fully dynamic environment as mentioned below :
  - It requires a parameter specifying the batch size.
  - It is forced to delete trained models to make room for new ones.
  - It cannot learn from the most recent examples until a new batch is full.

In both binary as well as multi-class classification, the procedure for inserting updates into the dynamic graph can vary. In the batch-based learning approach, data is streamed in the form of batches but without timestamps for every update inside the batch. The lack of timestamps leads to the inefficient operation of simultaneous deletion and addition operations inside the batch. Online and instance-incremental learning show that streaming data in line, update by update, is an effective way of continuously revising the fully-dynamic graph. Thus, this instance-incremental learning methodology has been incorporated into the framework described in this paper.

The classification model aims to incorporate multi-class classification in fully dynamic graphs using the instance-incremental learning methodology. The contribution of the Multi-Class Instance-Incremental (MCII) framework presented in this paper is therefore threefold. First, the framework includes a multi-class classification technique, with the nodes being classified into 'k' different classes as required. This is suitable for a wide range of applications where binary classification mechanisms would be insufficient. Second, the approach is targeted towards graphs that are fully dynamic in nature, i.e. graphs that can accept all kinds of updates (addition, deletion and modification) rather than just incremental updates which increases its applicability in real life scenarios. Third, the instance-incremental learning methodology is accommodated in the model for incorporating updates into dynamic graphs as a more efficient alternative to the batch-incremental approach. Additionally, since all historic data is being preserved in order to accommodate a fully dynamic graph as mentioned above, it serves a secondary function of increasing the accuracy of the classifier on graphs of all sizes since all past support vectors are being preserved. However, this might lead to increased space and time complexity in large scale graphs in particular.

The rest of the paper is divided into the following sections: In Section 2, the definitions and preliminaries adopted during the course of the paper are described. The mechanisms used in the process are also described in brief. The formal problem definition for the classification in fully dynamic graphs is also stated. Section 3 describes the framework adopted in detail. It includes a comparison with the existing framework in place and further includes a detailed description of the various algorithms used in the process. In Section 4, a description of the dataset used is included along with a detailed description of the experimental results obtained. Section 5 provides a look into the existing literature and related work. Finally, some conclusions are provided in Section 6 and the scope for future work is looked at in Section 7.

2 Preliminaries and Problem Formulation

In this section, we define certain key notations and definitions that will be used throughout the paper and thereafter present the problem definition.

2.1 Preliminaries

In this paper, the focus is on fully dynamic graphs, i.e. graphs that are subjected to addition, deletion as well as modification in the form of a continuous stream of updates. Each new node in the graph will be classified into a set of predefined classes. $X = \{X_1, X_2...X_n\}$ denotes the multiple labels assigned to the graph.

Definition 1 (Graph/Network): A labeled graph/network is denoted as $G = (V, E, X)$, where V is a set of vertices or nodes in the graph, $V = \{v_1, v_2, , v_n\}$, $E\epsilon(V * V)$ is a set of undirected/ directed edges between any two vertices and X is a set of labels that can be assigned to each node to identify their unique characteristics.

In order to extract the informative nodes from the graph/network, subgraphs are extracted which discard the less informative nodes. They are defined as follows:

Definition 2 (Subgraph): Let $G = (V, E, X)$ and $G' = (V', E', X')$ be two labeled graphs. G' is said to be a subgraph of G, i.e., $G' \subset G$, if and only if (1) $V' \subset V, (2)E' \subset E, and (3)X' \subset X$. Alternatively, if $G' \subset G$, then there exists a one-to-one (Injective) mapping between $V'\&V; E'\&EandX'\&X$. Also, if the above statement holds, then G is a supergraph of G'.

The subgraph extraction algorithm is called for every new major node added to the graph in order to extract

the informative minor nodes in its subgraph. The two kinds of nodes can be defined as follows:

Definition 3 (major & minor nodes): A node $v_M \epsilon V$ in the graph $G(V, E)$, where V is the set of nodes and E is the set of edges in graph, is classified as a major node if it corresponds to the type of data being classified. These major nodes have a corresponding class label. All other nodes that are utilized in assisting the prediction of the major node are termed as minor nodes(v_m). A major node v_M can be connected to another major node v_M or a minor node v_m but there exists no edge such that the two nodes corresponding to the edge are both minor nodes.

As mentioned before, this framework is aimed at graphs that are fully dynamic in nature and can accept all kinds of updates. Such a graph and its corresponding updates are now defined.

Definition 4 (Dynamic Graph): A fully dynamic graph is defined over a sequence of regular updates $(..U_i, U_i + 1, ..)$ where each update can be either an addition, deletion or modification operation that is to be applied to the existing graph in a continuous, streaming fashion.

Definition 5 (Update): An update, denoted by U, can be of three types: (i) U_a which inserts nodes and edges into the existing graph, (ii) U_d which deletes nodes and edges from the existing graph and (iii) U_m which modifies nodes and edges in the existing graph. In this paper, modification is treated as a sequence of deletion and addition updates in that fixed order, i.e. $U_m = U_d + U_a$.

Each update is denoted by $U(T, S, Node/Edge, X)$ where T signifies the timestamp of the update, S denotes the type of the update, Node/Edge indicates the node to be added to or deleted from the graph, and X represents the corresponding node label of the node to be updated in the current update. Each update streams into the graph individually according to their timestamps, i.e. the earliest timestamp will be the first update and so on. S or type of update is a set of binary values 0 & 1, where 0 represents the update of the type addition (U_a), and 1 represents the update of the type deletion (U_d). Updates involving modification (U_m) will be incorporated into the MCII framework over two consecutive updates as a combination of deletion (U_d) of the node followed by addition (U_a) of the new/modified node in this fixed order.

The framework described in this paper uses the instance-incremental approach for incorporating updates into the dynamic graph.

Definition 6: (Instance-Incremental Learning): The instance-incremental learning method (Read et al., 2012) involves learning from each individual, new update as they are incrementally added to the graph under consideration. The learning model is updated as soon as a new learning example is added to the graph.

This approach also makes use of graph kernels, namely the W-L kernel as part of the process. This kernel is responsible for calculating the similarity measure between the graphs. The W-L kernel is now discussed below.

Definition 7: (W-L Kernel): Graph Kernels are kernel functions on pairs of graphs that are intuitively, a measure of similarity between those pairs. This similarity is articulated by computing the inner products on these graphs (Vishwanathan et al., 2010). Weisfeiler Lehman (W-L) Graph Kernels (Shervashidze et al., 2011) are based on the 1-Dimensional variant of the Weisfeiler-Lehman Isomorphism test or the naive vertex refinement (Weisfeiler and Lehman, 1968). The W-L kernel represents the graph using a set of original and compressed node labels, which are generated after every iteration of the isomorphism test. The runtime complexity of the kernel is directly proportional to the number of edges in the graph and number of iterations of the isomorphism test.

2.2 Formulation of Problem Definition

Having stated the definitions mentioned in the previous subsection, we are now in a position to formally devise the following problem statement for the purpose of this paper:
Given a fully dynamic graph consisting of major and minor nodes in connection, with each major node being assigned one of 'n' possible class-node labels the goal is to design a learned classifier that is able to modify itself according to the type of update being inputted (addition U_a, deletion U_d or modification U_m) and thereafter to suitably predict new unclassified nodes into 'n' number of different classes.

With the problem statement and other definitions clear, Section 3 provides a detailed discussion on the working of the framework which aims to achieve the task described in the problem statement.

3 Framework

The Multi-Class Instance-Incremental Classification framework (MCII) proposed in the paper is explained using two subsections. First, an analysis with reference to the current work presented by (Yao and Holder, 2014) is done in order to explore the improvements suggested in this paper. Second, the key structural elements composing these improvements are described in detail.

3.1 Analysis in Reference to Existing Literature

This subsection provides a detailed description of the key structural changes in the MCII framework as compared to the (Yao and Holder, 2014) paper architecture. The framework accommodates all kinds of updates, including addition, deletion and modification. Thus, it effectively encompasses approaches that solely target incremental updates. (Yao and Holder, 2014) have incorporated a framework, considering only incremental updates, which uses a sliding-window to discard old data in order to conserve memory and to be applicable to large datasets. However, since the framework presented in this paper aims to incorporate updates in the form of deletion of nodes and edges as well, it preserves any and all historic data to produce a classifier that can successfully accommodate such updates. This eventually results in a highly accurate classifier.

In the MCII framework, represented by Figure 2, the data is first inputted into the existing graph using the instance-incremental format as opposed to accumulating consecutive updates into a batch input as done by the framework in (Yao and Holder, 2014), as shown in Figure 1. Secondly, the type of updates is not restricted to incremental updates alone as is the case in the previous approach, but also includes node removal and modification. Thirdly, the sliding window technique incorporated in the aforementioned literature has been removed, which allows for retention of all past inputted and processed data. Lastly, the MCII framework produces a multi-class classification model which also includes the binary classification model presented in Figure 1.

The subgraph extraction routine is a modification of the technique presented in the framework of (Yao and Holder, 2014). It has been extended to include deletion and modification updates with the routine also extracting the relevant subgraph around the node to be removed. The other major modification is the adaptation in order to accommodate the multi-class classification structure as well. The entire modified subgraph extraction routine is described below.

Subgraph Extraction: For extracting subgraphs, an entropy-based extraction routine inspired from (Yao and Holder, 2014) is presented in order to allow for multiple classes of classification. This technique assists in extracting a subgraph consisting of the informative nodes neighboring the node being classified and discarding the ones with less discriminative power. This is done by calculating the entropy value for each of the neighboring minor nodes corresponding to a major node and filtering out the less informative ones using a threshold measure θ (theta). The lower the entropy of the node, the more informative it is. We build on the work of (Yao and Holder, 2014) by extending this measure to include more than two classes as shown below.

For node $V_i \epsilon G$, if it is connected to any major nodes, we can define the entropy value for V_i. Let $N(V_i)$ be the neighbor nodes of V_i, and let $n_1, n_2, n_3 ... n_k$ denote the number of major nodes belonging to the k different classes. The probabilities of the instances of the k classes in $N(V_i)$ can be calculated in the following manner:

$$P_1 = \frac{n_1}{\left(\sum_{i=1}^{k} n_i\right)};\ P_2 = \frac{n_2}{\left(\sum_{i=1}^{k} n_i\right)};$$

$$P_3 = \frac{n_3}{\left(\sum_{i=1}^{k} n_i\right)};\ P_4 = \frac{n_4}{\left(\sum_{i=1}^{k} n_i\right)};\ ...$$

$$\text{In general, } P_k = \frac{n_k}{\left(\sum_{i=1}^{k} n_i\right)} \qquad \text{Equation(1)}$$

Thus the entropy computation for V_i using the generalized form of probability in Equation 1, can be denoted as follows:

$$E(V_i) = -\sum_{i=1}^{k} (P_i \log_2 P_i) \qquad \text{Equation(2)}$$

Only those nodes are included in the subgraph that have an entropy value less than or equal to the predetermined threshold. This process continues till another major node is encountered. The subgraphs obtained through this procedure merge with the retained SVM data to form the training data for the newly updated graph.

3.2 Methodology

This section aims to give a detailed understanding of the working and structure of the MCII framework proposed in the paper.

The MCII framework broadly operates using the following six steps:

1. Input - New updates are added using the instance-incremental method, i.e. one update per timestamp in the graph.
2. Driver - Once an update in entered, an outer envelope algorithm is called to determine if it is an addition, deletion or modification update.
3. Subgraph Extraction - For the addition update, a subgraph extraction routine is called in case the added node is a major node.
4. Training Set - The resulting support vectors are combined with the ones obtained previously to form the new training set which is used to produce the new classification model.
5. Deletion - In case of a deletion update, connections to the corresponding major or minor node are removed from the graph and the support vectors previously stored are suitably modified. A new

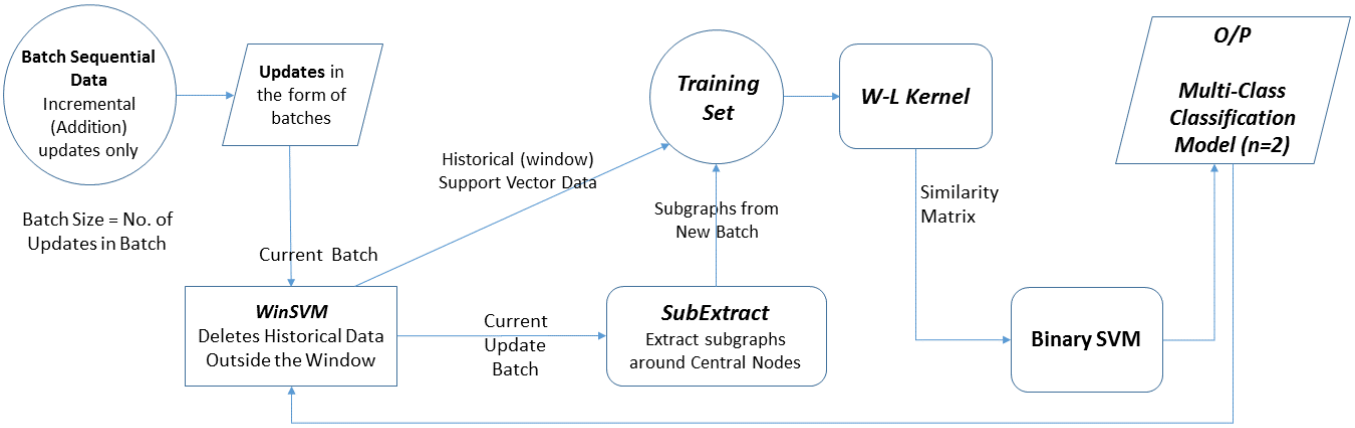

Figure 1 (Yao and Holder, 2014) Framework

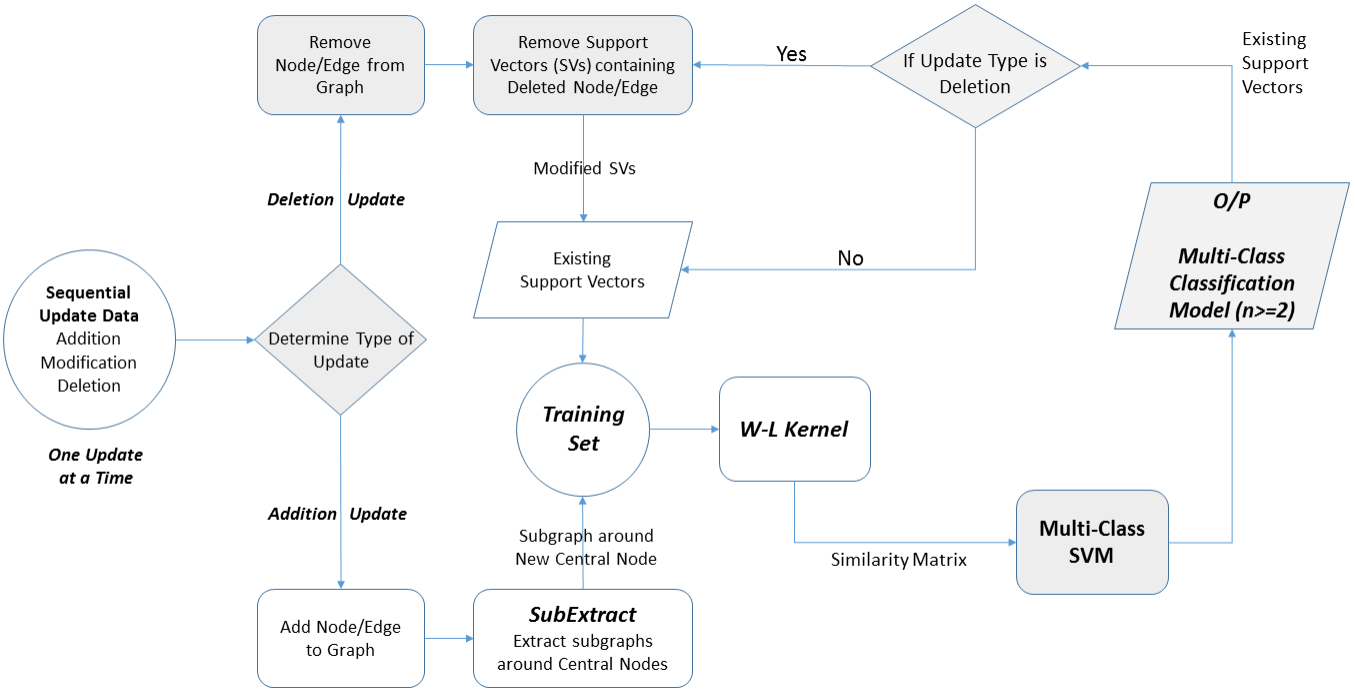

Figure 2 MCII Framework with augmented functionalities highlighted.

Algorithm 1: Driver (G, U(t), T, RetainedSV)

An envelope algorithm that takes as input an update and calls the necessary function according to the nature of the update.

Input	Output
G: A graph ***RetainedSV***: A set of support vectors retained till time (t-1) ***U(t)***: Update operation at time t ***T***: Entropy threshold	***Model***: Multi-Class classification model

```
1: if U(t) is delete operation then
2:     DelNode = node in update U(t)
3:     Model(T) = call DynamicDel(G, RetainedSV, DelNode, T)
4: else if U(t) is addition operation then
5:     AddNode = node in update U(t)
6:     Model(T) = call DynamicAdd(G, RetainedSV, AddNode, T)
7: end if
8:     return Model(T)
```

Figure 3 Algorithm: Driver

Algorithm 2: DynamicAdd (G, RetainedSV, AddNode, T)

A sub-algorithm corresponding to the addition update. This algorithm modifies the training set according to the new node being added and learns a classifier using the updated training data.

Input	Output
G: A graph ***RetainedSV***: A set of support vectors retained till time (t-1) ***AddNode***: Node to be added at time t ***T***: Entropy threshold	***Model***: Multi-Class classification model

```
1:  TrainData td = RetainedSV
2:  if AddNode is a central node then
3:      newSubGraph = computeSubGraph(G, AddNode, T)
4:      td = td + newSubGraph
5:      compute similarity matrix between the existing and updated graph using W-L kernel
6:      Model(T) = learn SVM classifier using the computed kernel matrix
7:      return Model(T)
8:  else if AddNode is a minor node then
9:      add AddNode to graph G
10:     return modified Model(t-1)
```

Figure 4 Algorithm: DynamicAdd

Algorithm 3: DynamicDel (G, RetainedSV, DelNode, T)

A sub-algorithm corresponding to the deletion update. This algorithm modifies the training set according to the new node being deleted and learns a classifier using the updated training data.

Input	Output
G: A graph ***RetainedSV***: A set of support vectors retained till time (t-1) ***DelNode***: Node to be added at time t ***T***: Entropy threshold	***Model***: Multi-Class classification model

```
1:  TrainData td = RetainedSV
2:  if DelNode is a central node then
3:      newSubGraph = computeSubGraph (G, DelNode, T)
4:      td = td - newSubGraph
5:      modify Support Vectors in RetainedSV which contain DelNode
6:      Model(T) = learn SVM classifier based on training set td using W-L kernel
7:      return Model(T)
8:  else if DelNode is a minor node then
9:      delete DelNode from graph G
10:     return modified Model(t-1)
```

Figure 5 Algorithm: DynamicDel

classification model is now learnt using the new training set.

6. Modification - In case of a modification update, a fixed sequence of the steps mentioned for the addition and deletion updates is followed.

After introducing a brief overview of the framework above, we now elaborate on the three key elements that form the basis of the structure for the MCII framework. These components are described in detail in the following subsections.

3.2.1 Targeting Fully Dynamic Graphs

The scope of the framework lies in working with datasets formed by continuous streaming data modeled as graphs, i.e. dynamic graphs. Dynamic graphs incorporating the following three types of updates are termed as fully dynamic graphs:

1. *Addition* - This type of update involves the addition of new nodes, edges or both into the existing graph data set.

2. *Deletion* - This type of update refers to the removal of nodes from the existing graph.

3. *Modification* - This type of update involves modifying the label of a particular node or changing the connections it has between the node itself and other neighboring nodes. Modification can be essentially viewed as a deletion plus addition function in that fixed sequence.

Working with this graph datasets requires an algorithmic setup that can suitably and accurately deal with all kinds of updates and learn a classifier model that can give sufficiently accurate results. The aim of the framework is to thus, learn a classifier that can efficiently work with a fully dynamic graph accepting the above mentioned updates. These updates are incorporated into this framework using the following algorithms.

The framework initially calls the Driver algorithm shown in Figure 3. The Driver algorithm is the outer envelope of the proposed framework. The function of the Driver algorithm is to determine the nature of the input update and to accordingly call the necessary function in order train the data. The updates are inputted to the Driver algorithm using the instance-incremental learning method which has been described in detail in Section 3.2.3. The other inputs to the algorithm include the dynamic graph being classified (G), a set of the support vectors retained till the previous time instant (RetainedSV) and the entropy threshold used for subgraph extraction (T). We test this framework with varying values of T as shown in Section 4. The DynamicAdd algorithm shown in Figure 4 is used when the update being entered is of the incremental nature, i.e, a node or edge being added to the existing graph. The DynamicDel algorithm shown in Figure 5 is used when the update being entered refers to a node or edge being deleted from the existing graph.

3.2.2 Multi-Class Classification

As mentioned in Section 1, a key idea of this paper is to incorporate the aspect of multi-class classification in the domain of fully dynamic graphs. This type of classification provides a more accurate determination of the class to which a particular unclassified instance

belongs to. To accommodate this multi-class aspect, the MCII framework uses the one-vs-all approach in order to extend the traditional binary classification to accommodate multiple classes. This approach involves one binary classifier per class that takes the instances of that class as positive and all other classes combined as negative. This series of binary classifiers results in an efficient multi-class classifier. This paper considers binary classification to be a subset of multi-class classification with the number of classes restricted to two in such a case. Hence, a multi-class approach will result in a more efficient classifier for all purposes.

3.2.3 Instance-Incremental Method of Updation

As mentioned in Sections 1 and 2, this approach makes use of the instance-incremental learning method for incorporating updates in the framework. This means that a single update will consist of the addition or deletion of only a single node in the graph. This serves the following advantages when applied to MCII framework:

- It incorporates each update as and when it arrives without the need to wait for a fixed number of updates that would constitute a batch. Hence, it is truly incremental in nature.
- This in-turn eliminates the need for intermediary buffers to store these batches, hence reducing the memory requirement for the framework.
- Since the MCII framework is dealing with deletion and a two-update modification as well, this approach preserves the correct sequence of updates in these two cases.

For the correct classification of fully-dynamic graphs and to reduce unnecessary memory usage, instance-incremental learning is the most effective method.

After these theoretical details, a practical working of the MCII algorithm and its implementation over various cases has been discussed in Section 4.

4 Experimentation

The MCII framework proposed in this paper is now tested by conducting multiple experiments on a sample dynamic dataset. This section tests the ability of the MCII framework to deal with fully-dynamic networks of varying specifications. In order to do so, a wide range of case studies are performed on the real world dynamic-graph dataset.

4.1 Experimental Setup

The computation of kernel matrix values and subsequent SVM classification was done under the MATLAB setup. The SVM functions are performed using the LibSVM library for MATLAB (Chang and Lin, 2011). A few constant parameters are used throughout the case studies performed. The maximum number of iterations in the W-L kernel isomorphism test is set at 5, keeping it consistent with the work presented in (Yao and Holder, 2014). The range of the threshold parameter theta is from 0 to 1, 0 being the case where no neighbor nodes are taken into the subgraph and 1 being the case when all the neighbors are included in the extracted subgraph. For testing purposes, we have varied theta in the range of 0.25 to 1 with increments of 0.05 as the level of information extracted from the neighbor nodes can be considered negligible for lower threshold values. The number of updates in a single iteration of the algorithm is fixed at 1, following the instance-incremental method of incorporating inputs. The machine used has a 2.7 GHz, Intel core i5 processor with 8GB RAM running Windows 7 Professional Operating System.

4.2 Dataset Details

The MovieLens dataset generated by GroupLens Research Group (Harper and Konstan, 2016), relates movies and users through the ratings given in order to indicate their liking towards a certain movie. The data was collected from the movielens.org website, which is a movie recommendation service website. Each movie in the dataset is related to users, their ratings, tags and genres. The dataset used for experimentation purposes has 100,000 ratings given to 10,000 movies by 700 users. . The movie IDs and user IDs are represented as nodes and the ratings between them form the edges. In this representation, the movie IDs denote the major nodes and the associated user IDs represent the minor nodes. There is an edge connecting a minor node to a major node if a user has rated that particular movie. This also indicates that the user has viewed that particular movie, i.e., the data is generated with the assumption that the ratings are given only after the movie has been seen by the viewers. The graph used, is a network of 10997 nodes and 105339 edges.

The multi-class classification task targeted here is the prediction of the genre-classes for the set of movies. The goal of the experiment is to learn a new classifier model iteratively after each update and check the accuracy for prediction of the genre (i.e. class) of the movie nodes.

We analyze this dynamic network by taking four case studies. For the deletion and modification case studies, we have modified the existing MovieLens dataset in order to accommodate these kinds of inputs to the dynamic graph. An important point to be considered here is that the optimum threshold and highest accuracy are variables which may change from a case to case basis with any change in the dataset or the type of classification being undertaken.

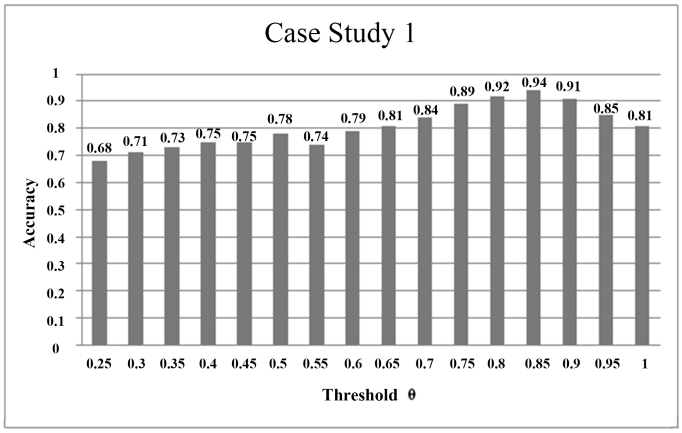

Figure 6 Case Study 1

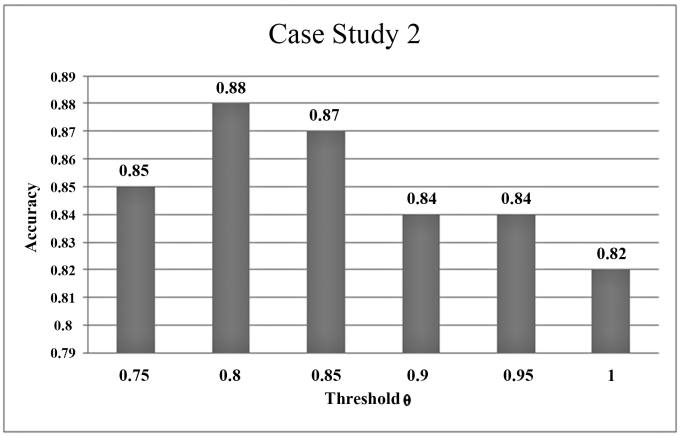

Figure 7 Case Study 2

4.3 Case Studies and Results

In order to exhaustively test the MCII framework proposed in this paper, we have considered four case scenarios where this framework can be applied.

4.3.1 Case Study 1

For the first case study, the data is divided into 3 major classes, containing the following genres:

- Class 1 - Adventure, Action, Thriller, Horror
- Class 2 - Comedy, Animation, Fantasy, Children
- Class 3 - Drama, Romance, Mystery and all remaining genres in the dataset

The dynamic graph is formed by new movies being constantly added to the network at regular intervals, and consequently, new edges being formed as users rate the movies. The purpose of the first case study is also to serve as a pre-testing or validation set for the second case study by obtaining the optimum parameters for this particular type of dataset and its unique features. The values obtained here will also be a healthy approximation of the optimal values for the fully dynamic nature of the graph, since we are dealing with all updates in an instance-incremental learning setting.

The incremental real world scenario tested in this case study deals with addition of only one new node or edge to the existing graph at any timestamp. This scenario was tested with 16 different values of to find its optimum value. At the optimum value of , the algorithm will be most efficient in terms of its accuracy. A peak value is obtained at the threshold, $\theta = 0.85$, as can be seen in Figure 6. It was also observed that the accuracy of the system increases from $\theta = 0.25$ to $\theta = 0.85$, but suffers a sudden decline from that point onwards till it reaches $\theta = 1.0$. This is the point where all the possible minor nodes, regardless of their entropy values, are added to the subgraph of the major node under consideration. The maximum value of the threshold is thus restricted to 1.0. This result allows us to focus on future case studies based on this dataset with an optimal θ value of approximately 0.85.

4.3.2 Case Study 2

We next observe the applicability of the proposed framework on the basic binary classification case in a fully dynamic graph with movie nodes being sporadically deleted as well. We clubbed the various classes present in the dataset into toe broad categories. The two classes are:

- Class 1 - Action, Animation, Comedy, Drama and Romance
- Class 2 - Adventure, Thriller, Mystery, Fantasy and other remaining genres

We observe that the MCII framework gives sufficiently accurate results as shown in Figure 7 for the binary classification case with the optimal value of the parameter θ being 0.85.

4.3.3 Case Study 3

For the third case study, the data is divided into the same three classes used in Case Study 1. However, while the first case considered incremental updates alone, we now study the situation where a node or edge may be deleted as well. To serve this purpose, the movie dataset is manipulated such that movies are getting deleted as well added to the graph at random time intervals. At any given timestamp, a single update is entered which can be of either the incremental or deletion kind. For a single deletion, all the elements in the row and column corresponding to that particular node in the graph matrix is equated to zero, thereby effectively removing all connections to the rest of the graph.

Taking motivation from the previous test cases, we can expect $\theta = 0.85$ to be the optimum case of maximum accuracy. This is verified as shown below by the experimental results. It can be seen in Figure 8 that

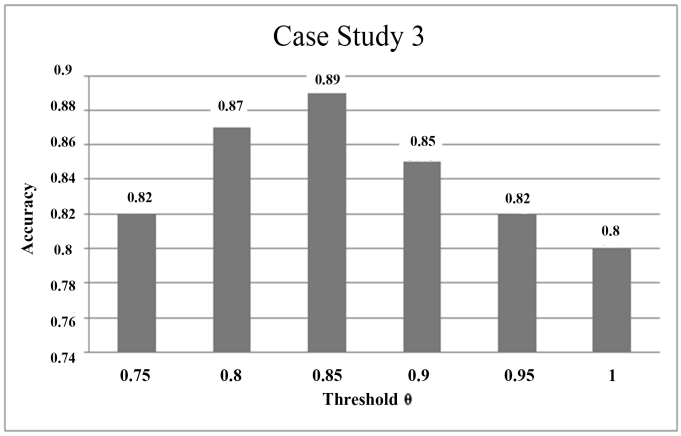

Figure 8 Case Study 3

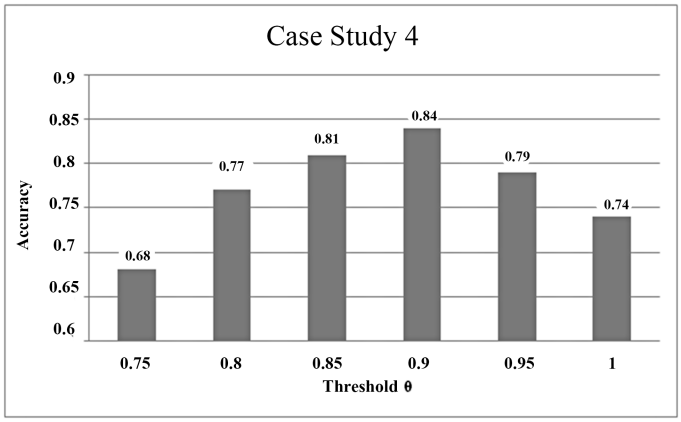

Figure 9 Case Study 4

the accuracy steadily increases till it attains its highest value of 0.89 at $\theta = 0.85$ and declines almost linearly following that.

4.3.4 Case Study 4

The fourth and final case study is a step towards testing the stability of the framework in a classification involving a large number of classes. For this purpose, the dataset is divided into six classes as follows:

- Class 1 - Action, Thriller
- Class 2 - Adventure, Children
- Class 3 - Comedy, Animation
- Class 4 - Drama, Fantasy
- Class 5 - Romance, Mystery
- Class 6 - Others

The same transformed dataset from case study three is used in this case as well. This case study is aimed at testing the performance of the framework under circumstances where a large number of classes are required in the process of classification.

It can be seen from the Figure 9 that the optimum threshold value changes as we increase the number of

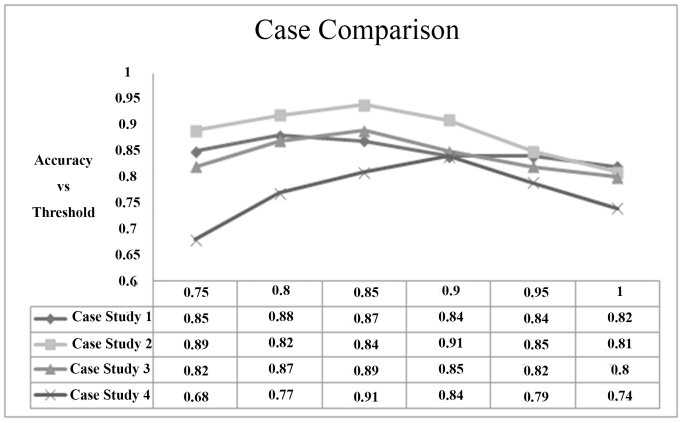

Figure 10 Comparison of Case Studies

classes from three to six from the previous case. As in the previous cases, we can observe the accuracy of the framework steadily increase as it approaches the optimum value of theta and then decline. The maximum accuracy obtained at $\theta = 0.9$ is 84%.

4.4 Comparison and Inference

The four cases have given us ample data based on the changes in accuracy plotted against the variable entropy threshold theta. Figure 10 represents a comparison among all the cases covered in this paper. The comparison chart shows that the optimum entropy threshold parameter (θ) and the corresponding accuracy value will vary on a case to case basis.

From the second case study, we observe that the proposed framework can be successfully applied to the conventional binary classification problem as well. Thus, binary classification problems effectively become a subset of the area of application of this framework. Using the data obtained from the case studies we also observe that the highest accuracy in each of the cases is above 80%. From the experiments conducted above, we can infer that on small to medium sized fully dynamic graphs, the framework gives an appreciable level of accuracy. The accuracy reaches its peak value at a particular value of the parameter theta. As shown by the above graph, this optimum value of theta varies on a case-to-case basis,i.e it changes with an increase or decrease in the number of classes in the classification model. The optimum value of this parameter is also dependent on the dataset being classified as is shown by the change in its value when we increase the number of nodes and edges or modify the dataset in any way. The MCII framework has succeeded in incorporating deletion updates and producing an accurate classification model, as evidenced by the second, third and fourth case studies. Hence, it is successfully applicable to fully dynamic graphs.

4.5 Validating the Model

To ensure the efficacy of the MCII framework, it undergoes a two step validation process. In the first step,

a k-fold cross validation is applied which is a technique widely researched upon in the past (Mosteller and Tukey, 1968, Mosteller and Wallace, 1963, Stone, 1974). It is used for selecting the optimal parameters required in the process of determining the classification model. In the second, a Macro-Averaged F-score is calculated after the execution of the framework has finished. These techniques have been elaborated upon in this section.

4.5.1 K-Fold Cross Validation

To better assess the results of the experimental analysis when generalized to an independent dataset, the MCII framework uses the k-fold cross validation. The algorithm performs a 10-fold validation process. The dataset is divided into ten equal-sized subsets out of which one is retained as validation data for testing and the remaining are used as training examples. The process is repeated ten times with each of the subsets performing the function of the validation set once.

The framework presented in this paper uses cross validation as a technique to obtain an accurate model of classification by selecting the optimal parameters (such as the cost function and gamma value for the kernel) for a support vector machine averaged over the 10 folds of the cross validation. This process also provides average cross-validation accuracy over the 10 iterations which is an indication of the correctness of the model generated.

4.5.2 Macro-Averaged F-Score

To evaluate the results obtained in a binary classification approach, the F-score (F) is one of the most common metrics used. It is the harmonic mean of Precision (Pr) and Recall (Re) values.

$$Pr_i = \frac{TP_i}{\left(TP_i + FP_i\right)} \qquad \text{Equation(3)}$$

$$Re_i = \frac{TP_i}{\left(TP_i + FN_i\right)} \qquad \text{Equation(4)}$$

Where, P_i, R_i = Precision, Recall of class i;
TP_i = Number of movies correctly assigned to class i;
FP_i = Number of movies assigned to class i that do not belong to that class;
TN_i = Number of movies that are not a part of class i and are correctly classified as not being part of that class;
FN_i = Number of movies not assigned to class i but are actually a part of that class;

These measures, Equations 3 and 4, for evaluating binary classification can be generalised for multi-class classification by using a Macro-Averaged F-Score measure. In macro averaging, the F-score is calculated individually for each class and then averaged over the number of classes. This can be mathematically shown as follows:

If the F-score of a class i is:

$$F_i = \frac{2 * Pr_i * Re_i}{\left(Pr_i + Re_i\right)} \qquad \text{Equation(5)}$$

Then the macro-averaged F-score over all the 'k' classes in the multi-class classification will be:

$$Macro - F = \frac{\sum_{i=1}^{k} F_i}{k} \qquad \text{Equation(6)}$$

Macro-averaging (Equation 6) gives equal weightage to all the classes in the classification.
Table 1 is a representation of the Macro-F scores and corresponding accuracy values for the two fully dynamic graph case studies.

The accuracies as well as F-scores obtained for the two case studies conclusively show the efficiency of the framework being presented.

5 Related Work

Graph classification has been a well researched area in the recent past, with domains like node classification being actively studied. (Aggarwal and Li, 2011) use node classification in the process of predicting class labels for specific nodes in the inputted dynamic graph. Another basis for distinguishing graph classification techniques is using the type of data the underlying graphs possess, static or dynamic. Significantly more work has been done on graphs containing static data (Deshpande et al., 2005, Gago-Alonso et al., 2013, Gärtner et al., 2003, Macskassy and Provost, 2007, Zhao et al., 2011) than a time-series based or dynamic data. The scope for improvement in the context of dynamic graph classification is thus immense with the existence of some literature in this area (Aggarwal, 2011, Aggarwal and Li, 2011, Chi et al., 2013, Yao and Holder, 2014, 2016). Out of these, (Aggarwal and Li, 2011) aim to combine content as well as the structure of the graph to obtain a classifier for the content-based dynamic network. The paper which comes closest to the work presented in the MCII framework is that of (Yao and Holder, 2014). It is one of the first contributions towards leveraging graph kernels to classify nodes inside a large-scale dynamic graph. They have constructed a model for classifying such dynamic graphs using a combination of Support Vector Machines and the Weisfeiler-Lehman graph kernel with special emphasis on targeting large-scale incremental dynamic graphs. The MCII framework discussed in this paper builds on the existing work by aiming to incorporate multi-class classification in fully dynamic graphs. The framework aims to address the limitations of the work presented by (Yao and Holder, 2014), making the classification model applicable to a wider range of problems. A major area of interest in the field of graph classification has been that of binary classification wherein the nodes are classified into two classes: positive and negative. There are several known binary classifiers in existence, Support Vector Machines (Hearst et al., 1998) and Regularised Least-Squares Classifiers (Rifkin et al., 2003) to name a few. (Yao and

Case	Optimum Threshold (θ)	Accuracy at Optimum Threshold (%)	Macro-F Score
Case Study 3	0.85	89	0.778
Case Study 4	0.90	84	0.734

Table 1 Macro-F Score Validation

Holder, 2014) provides a mechanism to apply the binary classification technique to large scale dynamic networks. With an increase in the complexity of data being managed, the need for efficient multi-class classification techniques has arisen. A survey of various multi-class methods of classification has been provided by (Aly, 2005). These include decision trees, neural networks, k-nearest neighbor and naive bayes classifiers. In this paper, we have used Support Vector Machines, one of the most robust techniques for classification. Our framework takes the traditional SVM used for binary classification and extends it to include multi-class classification using the one-vs-all technique (Rifkin and Klautau, 2004). This can then be decomposed to include binary classification as well. In the existing literature, there have been several approaches that use kernel-based methods for graph classification. Typically in these methods, a kernel function is devised that captures the graph similarity which is then used for classification purposes. In this paper, we use Support Vector Machines (SVM), one of the most well known classification technique which computes classifier models using input from graph kernels. For the purpose of this paper, we use a highly efficient Weisfeiler-Lehman graph kernel (Shervashidze et al., 2011). It makes use of the 1-dimensional variant of the Weisfeiler-Lehman Isomorphism test to compute similarity between 2 graphs. Another popular approach to graph classification includes frequent subgraph mining using SVMs (Yan and Han, 2002). The key idea is to identify the frequent subgraph occurring in the 'n' graphs of the example pool, construct feature vectors indicating the presence or absence of a particular subgraph and finally use SVMs to train a model to classify these vectors. Other methods of classification using SVMs include using the walk-based (direct product) kernel, random walks-based kernel and cycle based graph kernel (Ketkar et al., 2009). In this paper, we have extended the commonly used binary classifier to include more than two classes, i.e to allow for multi-class classification using the one-vs-all classification technique. We have also aimed to build on the existing work of (Yao and Holder, 2014) in order to make the multi-class classification model applicable to fully dynamic graphs in addition to that of solely incremental ones.

6 Conclusion

In this paper, we have presented a comprehensive, novel approach targeting multi-class classification in fully dynamic graphs. This has been achieved using a multi-class Support Vector Machine utilizing the one-vs-all technique in conjunction with the efficient Weisfeiler-Lehman (W-L) graph kernel. We have applied the instance-incremental method of incorporating the three different kinds of updates in order to accommodate all aspects of a fully dynamic graph. The framework also relies on a process of retaining maximum historical information at every iteration in order to increase the accuracy of the classification procedure. Additionally, a modified implementation of the subgraph extraction method for multiple classes has been incorporated into the framework to extract the most discriminative minor nodes for each of the major nodes under consideration. The efficiency of this integrated framework can be seen in the high accuracy rates and Macro-F scores obtained in different real-world classification scenarios. We have conclusively shown that this approach can be decomposed to apply to binary classification problems as well.

7 Scope for Future Work

The MCII framework presented in this paper has some scope for improvement in certain aspects. Since the framework retains the complete historic data, it ends up compromising on the time complexity of the system in order to gain higher accuracy. Conserving the entire history also results is greater memory usage. For both these reasons, although still retaining a high degree of accuracy, the framework could lead to an expensive algorithm in terms of the time and space complexity when the case of large scale graphs is considered. Our future work will focus on addressing these limitations. We hope to address the issue of large-size graph scalability while still retaining high accuracy.

References

Aggarwal, C. C. (2011), On classification of graph streams, *in* 'Proceedings of the 2011 SIAM International Conference on Data Mining', SIAM, pp. 652–663.

Aggarwal, C. C. and Li, N. (2011), On node classification in dynamic content-based networks, *in* 'Proceedings of the 2011 SIAM International Conference on Data Mining', SIAM, pp. 355–366.

Aly, M. (2005), 'Survey on multiclass classification methods', *Neural Netw* **19**.

Boulif, M. and Atif, K. (2015), 'Multi-objective cell formation with routing flexibility: a graph partitioning

approach', *International Journal of Computational Science and Engineering* **11**(2), 186–206.

Chang, C.-C. and Lin, C.-J. (2011), 'Libsvm: a library for support vector machines', *ACM transactions on intelligent systems and technology (TIST)* **2**(3), 27.

Chi, L., Li, B. and Zhu, X. (2013), Fast graph stream classification using discriminative clique hashing, *in* 'Pacific-Asia Conference on Knowledge Discovery and Data Mining', Springer, pp. 225–236.

Demetrescu, C., Finocchi, I. and Italiano, G. F. (2004), 'Dynamic graphs.'.

Deshpande, M., Kuramochi, M., Wale, N. and Karypis, G. (2005), 'Frequent substructure-based approaches for classifying chemical compounds', *IEEE Transactions on Knowledge and Data Engineering* **17**(8), 1036–1050.

Gago-Alonso, A., Muñoz-Briseño, A. and Acosta-Mendoza, N. (2013), 'A new proposal for graph classification using frequent geometric subgraphs', *Data & Knowledge Engineering* **87**, 243–257.

Gärtner, T., Flach, P. and Wrobel, S. (2003), 'On graph kernels: Hardness results and efficient alternatives', *Learning Theory and Kernel Machines* pp. 129–143.

Han, M., Duan, Z., Ai, C., Lybarger, F. W., Li, Y. and Bourgeois, A. G. (2017), 'Time constraint influence maximization algorithm in the age of big data', *International Journal of Computational Science and Engineering* **15**(3-4), 165–175.

Harchaoui, Z. and Bach, F. (2007), Image classification with segmentation graph kernels, *in* 'Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on', IEEE, pp. 1–8.

Harper, F. M. and Konstan, J. A. (2016), 'The movielens datasets: History and context', *ACM Transactions on Interactive Intelligent Systems (TiiS)* **5**(4), 19.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J. and Scholkopf, B. (1998), 'Support vector machines', *IEEE Intelligent Systems and their applications* **13**(4), 18–28.

Jin, N., Young, C. and Wang, W. (2009), Graph classification based on pattern co-occurrence, *in* 'Proceedings of the 18th ACM conference on Information and knowledge management', ACM, pp. 573–582.

Jin, N., Young, C. and Wang, W. (2010), Gaia: graph classification using evolutionary computation, *in* 'Proceedings of the 2010 ACM SIGMOD International Conference on Management of data', ACM, pp. 879–890.

Joachims, T., Hofmann, T., Yue, Y. and Yu, C.-N. (2009), 'Predicting structured objects with support vector machines', *Communications of the ACM* **52**(11), 97–104.

Ketkar, N. S., Holder, L. B. and Cook, D. J. (2009), Empirical comparison of graph classification algorithms, *in* 'Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on', IEEE, pp. 259–266.

Kong, X. and Philip, S. Y. (2014), Graph classification in heterogeneous networks, *in* 'Encyclopedia of Social Network Analysis and Mining', Springer, pp. 641–648.

Macskassy, S. A. and Provost, F. (2007), 'Classification in networked data: A toolkit and a univariate case study', *Journal of Machine Learning Research* **8**(May), 935–983.

Mosteller, F. and Tukey, J. W. (1968), 'Data analysis, including statistics', *Handbook of social psychology* **2**, 80–203.

Mosteller, F. and Wallace, D. L. (1963), 'Inference in an authorship problem: A comparative study of discrimination methods applied to the authorship of the disputed federalist papers', *Journal of the American Statistical Association* **58**(302), 275–309.

Pavlopoulos, G. A., Secrier, M., Moschopoulos, C. N., Soldatos, T. G., Kossida, S., Aerts, J., Schneider, R. and Bagos, P. G. (2011), 'Using graph theory to analyze biological networks', *BioData mining* **4**(1), 10.

Read, J., Bifet, A., Pfahringer, B. and Holmes, G. (2012), 'Batch-incremental versus instance-incremental learning in dynamic and evolving data', *Advances in Intelligent Data Analysis XI* pp. 313–323.

Rifkin, R. and Klautau, A. (2004), 'In defense of one-vs-all classification', *Journal of machine learning research* **5**(Jan), 101–141.

Rifkin, R., Yeo, G., Poggio, T. et al. (2003), 'Regularized least-squares classification', *Nato Science Series Sub Series III Computer and Systems Sciences* **190**, 131–154.

Rousseau, F., Kiagias, E. and Vazirgiannis, M. (2015), Text categorization as a graph classification problem., *in* 'ACL (1)', pp. 1702–1712.

Rytsareva, I., Chapman, T. and Kalyanaraman, A. (2014), 'Parallel algorithms for clustering biological graphs on distributed and shared memory architectures', *International Journal of High Performance Computing and Networking* **7**(4), 241–257.

Shervashidze, N. and Borgwardt, K. M. (2009), Fast subtree kernels on graphs, *in* 'Advances in neural information processing systems', pp. 1660–1668.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K. and Borgwardt, K. M. (2011), 'Weisfeiler-lehman graph kernels', *Journal of Machine Learning Research* **12**(Sep), 2539–2561.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K. and Borgwardt, K. (2009), Efficient graphlet kernels for large graph comparison, *in* 'Artificial Intelligence and Statistics', pp. 488–495.

Stone, M. (1974), 'Cross-validatory choice and assessment of statistical predictions', *Journal of the royal statistical society. Series B (Methodological)* pp. 111–147.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R. and Borgwardt, K. M. (2010), 'Graph kernels', *Journal of Machine Learning Research* **11**(Apr), 1201–1242.

Wang, T., Chen, Y., Zhang, Z., Xu, T., Jin, L., Hui, P., Deng, B. and Li, X. (2011), Understanding graph sampling algorithms for social network analysis, *in* 'Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on', IEEE, pp. 123–128.

Weisfeiler, B. and Lehman, A. (1968), 'A reduction of a graph to a canonical form and an algebra arising during this reduction', *Nauchno-Technicheskaya Informatsia* **2**(9), 12–16.

Yan, X. and Han, J. (2002), gspan: Graph-based substructure pattern mining, *in* 'Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on', IEEE, pp. 721–724.

Yao, Y. and Holder, L. (2014), Scalable svm-based classification in dynamic graphs, *in* 'Data Mining (ICDM), 2014 IEEE International Conference on', IEEE, pp. 650–659.

Yao, Y. and Holder, L. B. (2016), 'Incremental svm-based classification in dynamic streaming networks', *Intelligent Data Analysis* **20**(4), 825–852.

Zhao, Y., Kong, X. and Philip, S. Y. (2011), Positive and unlabeled learning for graph classification, *in* 'Data Mining (ICDM), 2011 IEEE 11th International Conference on', IEEE, pp. 962–971.

Zikopoulos, P., Eaton, C. et al. (2011), *Understanding big data: Analytics for enterprise class hadoop and streaming data*, McGraw-Hill Osborne Media.